OASIS Business Transactions Technical Committee
Transaction models sub-committee

Scope and Requirements, Actors & Terminology (incorporating Model Overview)

Alastair Green and Peter Furniss

2 May 2001

Status

This document is part of the on-going work of the OASIS Business Transaction Protocol Technical Committee.

It is a revised version of the Scope and Requirements, Actors & Terminology document from Alastair Green, dated 21. The document has been revised to incorporate the agreements reached at the 24 April BTP committee meeting in Wakefield (Boston). It incorporates material from Actors, Terminology and Transaction Model, Pal Takacsi-Nagy, 25 April and the summary of the conference call at the end of March on "four levels of thing".

Business Transaction Protocol

The name of the protocol is *business transaction protocol* or BTP abbreviated. The purpose of BTP is to orchestrate loosely coupled software services (e.g. web services) into a single business transaction. There are two kinds of business transactions: cohesive and atomic. The initial version of the standardised protocol focuses on atomic business transactions, but within a scope where they are components of cohesive business transactions. The transaction model and the actors that are described in this document are applicable to the atomic transactions.

Atomic business transactions

Atomic business transactions (sometimes just "atoms" below) are made up of services that all agree to enforce a common outcome of the transaction: in case of a failure all services un-do (compensate, roll-back) their operations that were invoked during the transaction, in case of a success all services make the results of their operation permanent. There is no assumption as to the mechanisms used by the services to achieve the un-do of the operations.

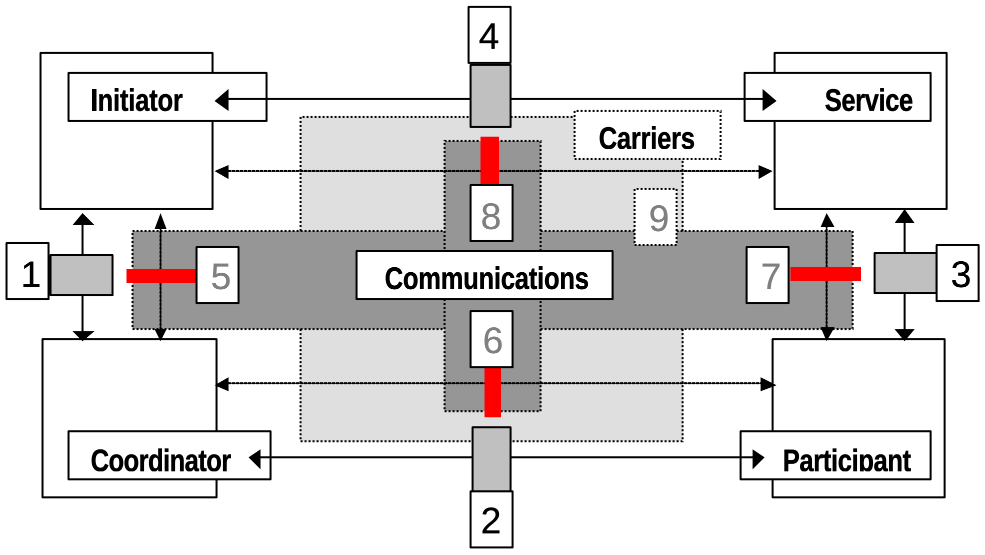
Cohesive business transactions

Cohesive business transactions (sometimes just "cohesions" below) are made up of several atomic transactions. The atoms forming a particular cohesion do not necessarily have a common outcome – under application control, some may be performed (confirmed), others may fail (cancelled - their operations are undone).

Note: The term "transaction" is used in a variety of senses in this area. Readers should avoid assuming properties of atomic business transactions and cohesive business transactions that are not specified as applicable.

Scope of the Specification

The scope of standardisation can be defined in terms of the possible actors involved. These can be crudely categorised by kind of software component – system or application; and by location/ownership – client/initiator or service. This gives four principal actors, as shown in the diagram (the detailed definition of the actors is given below)



Key:

1. Initiator/Coordinator (IC): the Demarcation API
2. Coordinator/Participant (CP): the Coordination Protocol
3. Participant/Service (PS): Participant API
4. Initiator/Service (IS): Operation Invocation Protocol

5. Identifier extraction: read context from Coordinator
6. CP communications: how the Coordination Protocol is carried
7. Identifier insertion: write context into Service/Participant
8. IS communications: how the Operation Invocations are carried
9. Communications/Carrier (CC): carrier bindings.

In terms of these interactions, the scope of the specification is

2, 6 - the coordination protocol itself, between coordinator and participant, with a specified binding to a carrier mechanism

5, 7 – the content of the context to be passed with or by the application messages

4 – the augmentation of the application messages with the context

The fact that the context must be passed with the application messages is essentially the responsibility of the application. Given a particular carrier (e.g. SOAP), the mechanism for carrying the context will be specified in a way that is general for all applications (c.f. implicit context propagation in OTS) – hence 4 is included.

Editors note: further editing needed on following sections PRF

The document will need to include an illustrative demarcation api (item 1 in the diagram), which will help ourselves discuss and readers understand. This would be clearly identified as non-mandatory – the intent is to explain the message set (2), not to standardise the api (1), though its presence should enhance the acceptability of the standard.

Other points:

There may be distinct entities that are the initiator, coordinator and decider (as in the HP input) – the decider involves application logic, though this may be handled by delegation to a decider with an appropriate fixed pattern.. This is essentially a matter of area 1 in the diagram.

It would be possible for a service/participant to also behave as an initiator/coordinator (forming a familiar tree). However, this is hidden from the perspective of our protocol – it is just the internal working of the participant, and a different context appears on the lower legs. In other cases, a web-service might be a simple "router" – the context would be passed unmodified, and registration/enlistment would by-pass the router entity.

Service description – there is a general need for a service to identify itself as supporting coordination, but the details are various (is a single flag enough, how does it fit with other work on service description/discovery).

This should be raised at the general meeting, seeking to get it considered by another subgroup.

However, it is desirable to include a mechanism to ensure that sent contexts aren't silently ignored if received by something that doesn't understand them (easy in SOAP, though not strictly achievable in our specification in general, since something that hasn't implemented our spec won't know that it must say it doesn't understand)

Requirements

Relationship to ACID properties:

*BTP (in both cohesive and atomic business transactions) will not **rely** on ACID properties, but will not **exclude** use and support of ACID properties. Note that*two-phase commit exchanges (between parties) do not require the use of two-phase locking (within the participant).

It should also be noted that:

> *The top of the tree may decide that all the bits of the business transaction will have the same decision (i.e. there is only one atom, or the cohesion determines that all atoms will have the same decision after all)*
>
> *The participants (in general) have the opportunity to apply isolation (and durability) mechanisms to the their data. (This allows a participant to use an XA resource manager, if it is desired and subject to operational considerations (such as duration of the transaction)*
>
> **If everyone agrees (or happens) to do this, then the cohesion is an ACID transaction.**
>
> *There will not be, within this protocol, mechanisms to negotiate or ensure this (this might be part of capability negotiation)*

Interoperability: the protocol shall support interoperation between web service implementations and user implementations using different system software

XML schema-based protocol message formats: the protocol messages shall be specified using XML schemas

"Resource" registration scheme: a means shall be provided such that resources (entities affected by the operation of the business transaction) can register as part of the transaction, without requiring prior knowledge of their existence and location at the initiator side.

> **Communications-protocol independence**: *Specifcation will define expectations on the underlying communcations, and give, but not mandate, a binding to some particular communication*

<u>**Deferred potential requirements**</u>

The items in this section are areas for possible future work.

Security requirements:

Configurable protocol authentication
Access control for joining and terminating transactions
non-repudiation of coordination messages

Status subject to security group deliberation, but cannot just pull in security mechanisms from elsewhere – there are specific issues here.

Operations can use (and can advertise) differing isolation levels (degrees of blocking)
Operations can use (and can advertise) differing durability levels (degrees of persistence)

Specification will not mandate any particular behaviour in respect of persistence and concurrency control for participants. (see requirement on ACID)

Application/operation negotiation over isolation and persistence levels.

Capability negotiation is deferred

Editors nots: Not sure what to do with the following section

Discussion on remaining requirement items was at model level rather than simple yes/no.

Requirement #1: **Operation groups with reverse operations**
Requirement #2: **Operation group atomicity**

Issues with who defines the group and interactions with concept of a web-service as a single concept – possibly they would always be groups with a single operation. There can be multiple registered participants in a group (possible from the same service entity)

The demarcator can put operations in the same group to indicate (and enforce) a tight relationship

Requirement #3: **Action demarcation (addition and removal of operation groups)**
Requirement #4: **Action reversal**
Requirement #5: **Multiple valid action outcomes**

Some felt the action (= cohesion) is at the collaboration level, controlled by workflow. Alternatively, the cohesion appears in the demarcation api (including the ability to choose partial outcomes and the ability to cancel the whole cohesion), but possibly is not explicit in the protocol.

Requirement #6: **Positive/negative action timeout**
Requirement #7: **Positive/negative operation timeout**

If a participant is able to unilaterally reverse or apply its vote, there needs to be a re-voting round, when it is asked what its current status is. (Such a re-vote can be regarded as the only vote, depending on assumptions about how the participants behave)

This re-vote does not need to involve all participants, if it is known which are liable to change state autonomously.

The re-vote needs to be certain – though this imposes timing constraints on the termination process.

The re-vote could be a query (that would not itself change the participant state), returned to the application that may then behave differently if some participants have made their own decisions.

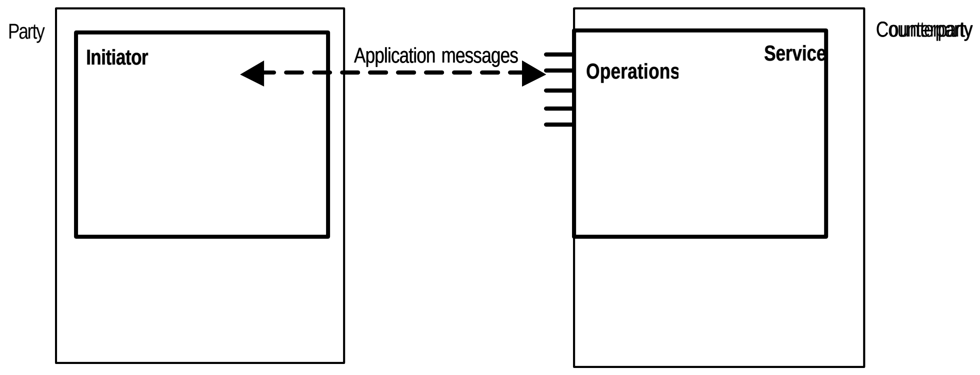
Actors & Terminology

Editors note: the original “Client” has been changed to “initiator” in this section.However the relationship between the application code that sends application messages with the (atomic) context, the code that triggers the termination/completion wave, the code that decides whether to confirm or cancel the (prepared) atom and the atom coordinator need some further defining.

The description that follows in the next three sections gives informal definitions of each term within the context of a general model. The section “Terminology—Formal definitions” gives more elaborate and precise definitions.

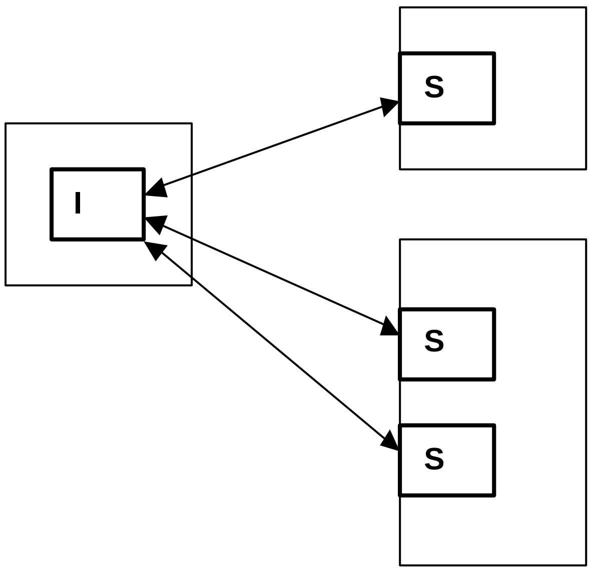
What is a business transaction? The application’s view.

First, we establish a very high level view of the actors, from an application standpoint.

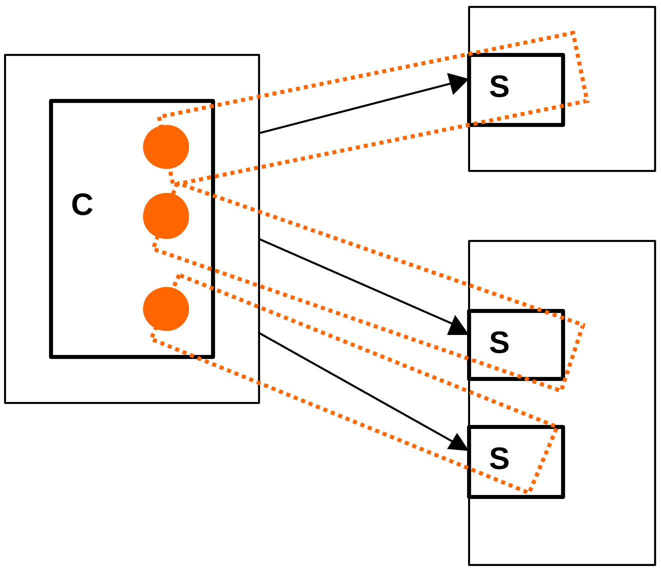


One organization (a **Party**) wants to use a web **Service** provided by another organization (its **Counterparty**). We define a service as a software agent which offers a clearly delimited set of **Operations** for invocation. The party’s **Initiator** application (a software agent) sends **Application Messages** to the service, in order to invoke operations. The service responds to the initiator in kind. Messages are communicated via **Carrier Protocols**.

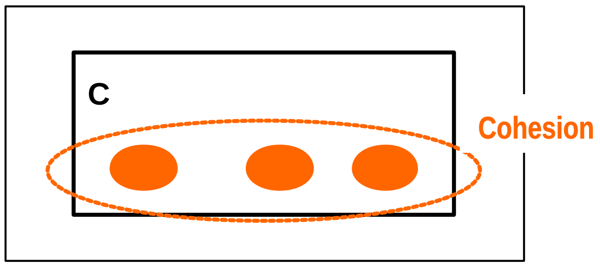
Of course, there might be several counterparties, and each of them might have more than one service to offer.



Taking it a step further, the initiator may use these services as atomic units of work. In other words, if the initiator invokes one or more operations from a particular service (e.g provisionalOrder, associatedShipping), it is highly likely to want these operations to succeed or fail as a unit. (We should be clear that atomic units of work might include operations from several services, even if that is less likely, albeit fully possible.) Making the simplistic assumption that the set of operations used by the initiator in each service constitutes an atomic unit of work (Atomic Business Transaction or **Atom**), we can recast the last diagram. The dotted frames represent these atoms. The blobs represent the initiator's view of the atom.



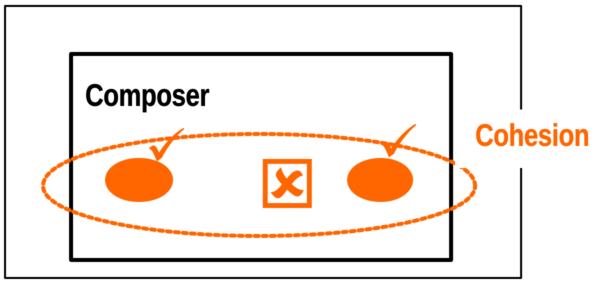
If we focus on the initiator side of this picture (and turn things through 90°) then we can perceive that the initiator has a set of atoms that it can manipulate. This set of atoms is known as a **Cohesive Business Transaction (or Cohesion):**



Imagine that the initiator can dictate whether each atom within the cohesion succeeds or fails, *even when it knows that the atom is capable of succeeding.* This is somewhat novel. In conventional transactional systems the initiator usually has a "demarcation" API which enables it to ask for an atom (an atomic action, a transaction) to "commit". This is shorthand for "ask a coordinator to gather the votes of all the participants that make up the atom, and then send commit messages to all participants if none voted to rollback, otherwise send rollback messages to all participants".

In the situation we are considering the decision as to the outcome of each atom may well require information about the readiness of other atoms. In other words, it would be wrong to allow each atom to commit autonomously, according to the views of its inferior participants. Instead we wish the initiator to act as "the last voter", who will instruct each atom to **Confirm** or to **Cancel**, in the full knowledge of the other voters' views, and in the light of the state of all other atoms that it thinks are relevant. (Note that this is allows any decision dependency graph to be constructed, and does not require the structural foreknowledge that is implied by nested or glued transaction models.)

To be more concrete: imagine that our three atoms are: order goods from a supplier; order shipping via the same supplier, and get a competitive quote for shipping from a third party. Once we know the cost of the two quotes, we can decide to confirm the cheapest (or most reliable, or best known, or biggest …) and cancel the other, while confirming the goods order itself. The final outcome of the cohesion is to accept two atoms, and reject one atom.



The cohesion is reduced, at the end of its life, to the two accepted atoms. In that sense we can view a cohesion as a structured means of whittling down the choices available to the initiator application, with the goal of deciding on a single, ultimately successful outcome. Alternatively, the initiator application, in this role as **Composer** of a cohesion's worth of atoms, may decide to cancel all of them, and abandon the attempted "business transaction" altogether.

The following code fragment illustrates how this might look to a Java initiator. Note that there is no need or intent to prescribe an API like this in our specification: the code is for illustrative purposes only (and has never been near a compiler!).

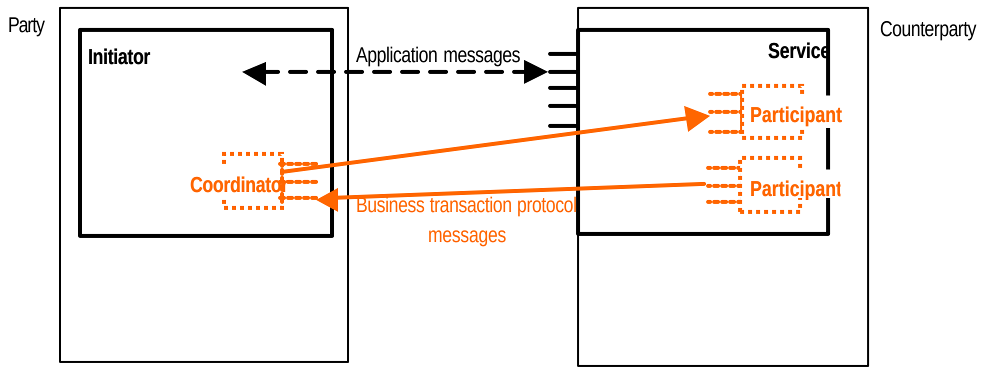
```
 void cohesionComposer() // an application method
{
     Atom orderGoods = new Atom();
     Atom shippingViaGoodsSupplier = new Atom();
     Atom shippingFromAnotherSource = newAtom();

     // application work

     Quote quoteForGoods =
         orderGoods.sendApplicationMessage (“quoteForGoods”, arg, arg …);
     Quote quoteForShippingViaGoodsSupplier =
         orderGoods.sendApplicationMessage (“quoteForShipping”, arg, arg …);
     Quote quoteForShippingFromAnotherSource =
         orderGoods.sendApplicationMessage (“quoteForShipping”, arg, arg …);

     // ensure that the quotes are guaranteed (may be folded into app messages)

     orderGoods.prepare();                // no exception, so it is ready
     shippingViaGoodsSupplier.prepare();  // ditto
     shippingFromAnotherSource.prepare(); // ditto

     // there are recovery subtleties that will require logging by the coordinators
     // after all the atom outcomes are decided, and before they are delivered:
     // see note on “Cohesion Outcomes” below . . .

     orderGoods.confirm();
     QuotesOutcome quotesOutcome
       = this.decideQuotesOutcome (quoteForShippingViaGoodsSupplier,
                                   quoteForShippingFromAnotherSource);
     quotesOutcome.selected().confirm();
     quotesOutcome.rejected().cancel();
  }
```

It should be noted that the cohesion may cancel atoms and create new ones during its lifetime, and that the membership of a cohesion is therefore established dynamically by the action of the applications. Atoms may also be cancelled "from below" by a participant of the atom. An example would be a service withdrawing its participants because of a timeout. A composer may take interim "polls" to discover whether atoms have gone ready or been cancelled, and is also able to decide that all atoms will finally be prepared in one sweep.

The Business Transaction Protocol per se

So far we have *assumed* that atoms can be brought to a state of readiness, where they are able to confirm or cancel (roll forward or backward); or that they are cancelled outright before reaching the state of readiness. The protocol used to achieve this is the standard two-phase commit protocol, familiar from classic transaction management.

The next, expanded diagram shows the significant actors involved in the atom outcome protocol. The protocol defines the content and sequence of messages that are sent between actors, and the contracts that determine their reactions.

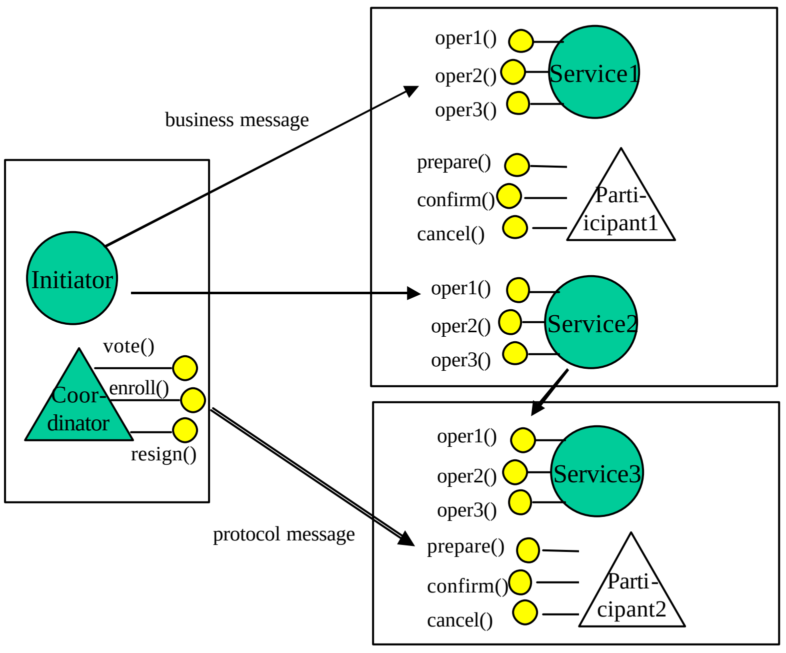


The counterparties have decided to make their services “BTP-capable”, and have somehow advertised this fact to the party (outside our scope). The initiator therefore decides to create (**Initiate**) an **Atomic Business Transaction** (not shown in the diagram), which means that a **Coordinator** is created to coordinate any **Participant**s that get involved in that atom.

The atom has an id (an **Atom Identifier**) which it piggybacks on an application message, and which the operation receives. If the operation wants to do some work which is capable of being undone by the atom cancelling then it **Enrols** a participant, which means that a message is sent back to the coordinator, telling it about the participant (which is identified by a **Participant Identifier)**. In the process of these exchanges both the coordinator and the participant get each other’s **Address**.

Any work that the service does which is related to this atom will be tagged with the participant id. (In fact, it may be convenient to group units of work into separate, multiple participants, which are used by the service and each of which is enlisted with the coordinator.)

The diagram below illustrates the transaction model for BTP:



At some point the initiator decides to **Terminate** the atom, which causes **Prepare** messages to all enrolled participants. The participant on receiving this message should log the information required to either **Confirm** or **Cancel** work done for this atom, so that it can either complete the work of the atom, or undo it. If it can do this, it sends a **VOTE/Ready** message to the coordinator; if it can't do this it sends a **VOTE/Cancel** message back. (The messages between the coordinator and the participant are **Business Transaction Protocol Messages**.)

If the coordinator receives any **VOTE/Cancel** messages then it sends a **CANCEL** message to all registered participants of the atom. Otherwise it waits to be told by the initiator whether to send a **CANCEL** or a **CONFIRM** to all of them. The participants do whatever makes sense to them, in either case. A cancel might reverse database changes, or do some other compensatory work that makes sense for the web service provider. The initiator is not aware of the details, but it may know that the **Contract** (legal or computerese) it has with the service implies certain things about a cancellation (like the web service won't go ahead with a credit card transaction).

Note that in most standard 2PC-based systems the coordinator automatically commits (confirms) if all the participants vote ready. Here we deliberately hand the decision up to the initiator. This allows the initiator to make complex decisions about the **Outcome** of the atom (confirm, cancel). These decisions are based on business rules and other (application-related) atom outcomes in a very plastic way. It is expected that the initiator will take a higher level, cohesion-style approach to coordinating the outcome of all of its work (involving many atoms), as discussed in the last section.

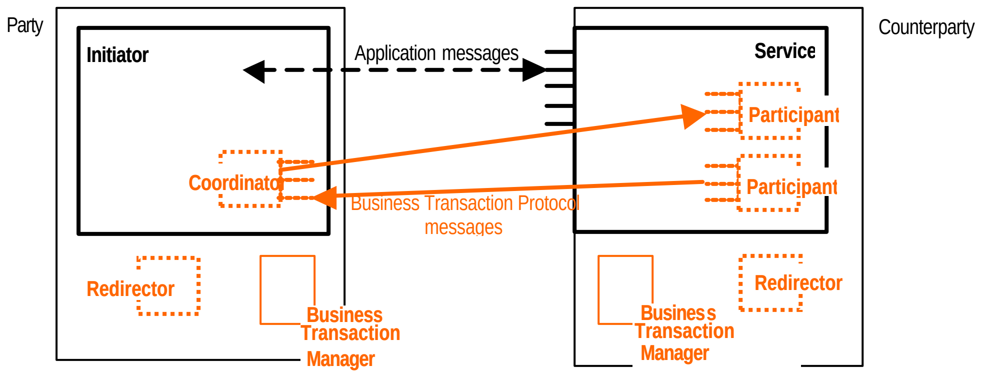
There are a couple of refinements that should be mentioned.

A web service is a highly autonomous actor, which can decide to withdraw or leave a cohesion. This implies that any participants it has enrolled may be asked to **Resign** from the atom they are enrolled with. This can only happen if they are **Ineffectual** (have made no significant changes). If they have made significant changes they must send an unsolicited **VOTE/Cancel**, which will cause the whole atom to be cancelled.
It is also possible for a participant to send an unsolicited **VOTE/Ready**.

A participant is free to act as a coordinator to some underlying participant(s). If this happens then we end up with a tree of participants, all of which are involved in the original atom. The combination of a participant and a coordinator is sometimes viewed as a "sub-coordinator", although there is no need for a special actor to be defined to allow this double role to be performed.

Crash Recovery and Addressing

A final extension of the diagram used previously shows an actor called a **Business Transaction** Manager, and one called a Redirector.



It is always possible that a receiver for a business transaction protocol message may be transient, and that a replacement endpoint may be put in its place. If that happens then the sender can try to obtain a replacement address, a facility which is provided by a redirector. A redirector allows transient actors to hand over their responsibilities. Coordinators and participants tell each other about their redirector(s). This preserves the autonomy of parties and counterparties.

When a business transaction protocol endpoint wants to communicate it uses a compound **Business Transaction Protocol Address**. The first part is meaningful for the carrier protocol being used for communication between the two endpoints. (i.e. it will permit a message to be delivered to a receiver). The second part is an opaque suffix, which might be bound to a refined address, or to an opaque appendix of a URL, or to a header for a particular carrier protocol. This may be used to route a message from a listener to the final, intended receiver. If an address is invalid then redirection should always be able to ultimately yield the address of a **Business Transaction Manager**. This is a backstop receiver which can respond "addressee unknown here". Without this the sender will not be able to distinguish between a communications failure and the absence of a receiver. In certain crash recovery scenarios the absence of a receiver enables conclusions to be drawn, whereas inability to transmit has to cause retries. It is necessary to provide a means to distinguish these two circumstances.

Terminology—Formal definitions

Actor	An entity which executes procedures, a software agent.
Address	An identifier for an *endpoint.*
Application	An actor which uses the Business Transaction Protocol.
Application Endpoint	An *endpoint* of an *application message*.
Application Message	A message produced by an *application* and consumed by an *application.*
Application Operation	An operation which is started when an application message arrives.
Appropriate	In accordance with a pertinent contract.
Atomic Business Transaction, Atom	A set of participants (which may have only one member), all of which will receive instructions that will result in a homogeneous *outcome*. (Transitively, a set of operations, whose *effect* is capable of *countereffect*.) An atom is identified by an **Atom Identifier** (a globally unique identifier).
BTP-aware service	A service that: • recognizes the business transaction context inside/received with request messages • triggers the enrollment of a Participant with the Coordinator, if the work done by its operation(s) should be part of the business transaction • A BTP-aware service may also propagate the transactional context to subsequent service operation invocations (Note: that both the Service and the Initiator actors have the inherent role of propagator)
Business transaction context	A data structure propagated onto messages passed between the initiator and the services within a business transaction. The Business Transaction Context is

	transaction. The Business Transaction Context is unique to a transaction instance and it contains information that helps the actors the execute BTP
Business Transaction Protocol Address	A compound address consisting of a mandatory *carrier protocol address* and an optional opaque suffix.
Cancel	Process a *countereffect* for the current *effect* of a set of procedures.
Carrier Protocol	A protocol which defines how *transmissions* occur.
Carrier Protocol Address	The address of an endpoint for a particular carrier protocol.
Cohesive Business Transaction, Cohesion	A set of Atomic Business Transactions for which the decisions are coordinated, though not necessarily all the same. All can be cancelled as a unit (the performance of cancellation being delegated in each atom)
Collaboration	A composite of activities, including cohesive and atomic business transactions that has some overall unity of application purpose, but where the handling of unsuccessful and cancelled parts can involve completely new, forward operations. (BTP does not directly support collaboration; use of cohesive business transactions may be in the context of a collaboration; alternatively a collaboration approach may be used instead of BTP.)
Confirm	Ensure that the *effect* is *completed* of a set of procedures.
Contract	Any rule, agreement or promise which constrains an actor's behaviour and is known to any other actor, and upon which any other knowing actor may rely.
Coordinator	An actor which decides the *outcome* of a single *atom*, and has a lifetime which is coincident with that of the *atom*. A conductor holds knowledge of the set of enrolled participants in the atom. A coordinator can issue instructions to a *participant* to *prepare*, *cancel* and *confirm*. These instructions take the form of *business transaction protocol messages*. A coordinator is identified by its *atom*'s *atom identifier*. A coordinator must also have a *business transaction protocol address* to which participants can send

	business transaction protocol messages.
Countereffect	An *appropriate* effect intended to counteract a prior effect.
Countereffect Contract	The contract which governs the relationship between the *effect* and the *countereffect* of a procedure. In the absence of any other overriding contracts the *countereffect contract* is the promise that "The *countereffect* will attempt so far as is possible to reverse or cancel the *effect* such that an observer (on completion of the countereffect) is unaware that the *effect* ever occurred, but this attempt cannot be guaranteed to succeed". Note that exact meaning of the countereffect depends on the implementation – there may be indirect or consequential effects of the original procedure that are not reversed.
Effect	The changes induced by the incomplete or complete processing of a set of procedures by an actor, which are observable by another contemporary or future actor, and which are made in conformance with a contract known to any such observer.
Endpoint	A sender or receiver.
Inappropriate	In violation of a pertinent contract.
Ineffectual	Describes a set of procedures which has no *effect.*
Initiator	An actor that starts a business transaction by invoking the Coordinator. The Initiator also sends messages or invokes operations on other services with the transaction context propagated onto the request message.
Message	A datum which is produced and then consumed.
Operation	A procedure which is started by a receiver when a message arrives.
Outcome	A decision to either *cancel* or *confirm*.
Participant	A set of procedures which is capable of receiving instructions from a *coordinator* to *prepare*, *cancel* and

	confirm (the termination protocol). A participant must also have a *business transaction protocol address* to which these instructions will be delivered, in the form of *business transaction protocol messages.* A participant is identified by a **Participant Identifier** (a globally unique identifier). A participant executes the termination protocol on behalf of a set of services that are associated with it. A participant is responsible for • deciding the timing of making the results durable (immediately, after timed period, when notified of successful termination) • making the results of all the operations of associated services invoked inside the transaction permanent if terminated with success, • or to execute compensating operations for all associated services if the business transaction terminates with failure.
Participant Identifier	A globally unique identifier assigned to a *participant*.
Prepare	Ensure that of a set of procedures is capable of being successfully instructed to *cancel* or to *confirm*.
Receiver	The consumer of a *message*.
Sender	The producer of a *message*.
Service	An actor which on receipt of an *application messages* may start an *application operation* which is *appropriate*. For example, a process which advertizes an interface allowing defined RPCs to be invoked by a remote initiator.
Transmission	The passage of a *message* from a sender to a receiver.

Business transaction protocol: Abstract message set

Context

An application message which communicates an operation of an atom from the initiator to the service is "augmented" with a context.

Parameter	
Atom identifier	
Coordinator address	

Meaning:

If there are changes from operations induced by the receipt of this message, these changes are to be subject to the decision of the atom. If this atom is unknown to the service receiving an augmented message, the ENLIST message shall be sent to the coordinator.

Comment on augmentation:

"Carrying" the context can be achieved in several ways – it may be in a header/envelope, or in a separate message on the same connection, among other means.

It is possible for a responder to pass on the context in a further message to some other entity – this can occur whether or not the first responder is itself registered (c.f. transactional server in OTS).

Protocol Messages

ENROLL

Sent from a participant to the coordinator (using the address in a received context)

Parameter	
Atom identifier	
Participant address	
Response_requested	yes/no

Meaning:

Sender wants to be a participant in this atom

The participant address is always needed, even when this is sent on a connection of some kind – the connection would allow the register-reply to come back, but may not still exist when the later messages are sent.

This message can be piggy-backed on an application message (typically on an application reply).

Response_requested is set to “yes” if an ENROLLED response is required.

The ENROLLED response will be necessary in the following cases:

a) Where the receipt of a PREPARE message has triggered processing that requires the enrolment of another participant – the first participant cannot send a VOTE until it can be sure the new participant is enrolled.

ENROLLED

Sent from coordinator to participant that has just sent a ENLIST with a Response_requested value “yes”.

Parameter	
Atom identifier	

Meaning:

The participant is enrolled in the atom – termination messages will be sent to it.

RESIGN

Sent from a participant to the coordinator if the operations of the atom have had no effect

Parameter	
Atom identifier	
Participant address	
Response requested	

Meaning:

The operations had no effect; the sender is no longer to be a participant in this atom.

Response_requested is set to “yes” if a RESIGNED response is required. It can be sent as a response to PREPARE, instead of VOTE.

RESIGN is equivalent to readonly vote in some other protocols, but can be issued early. The RESIGNED response will be needed if no PREPARE has been received, to ensure

RESIGNED

Sent from coordinator to participant that has just sent a RESIGN with a Response_requested value “yes”.

Parameter	
Atom identifier	

Meaning:

The participant is enrolled in the atom – termination messages will be sent to it.

PREPARE

Sent from coordinator to participant

Parameter	
Atom identifier	

Meaning:

Determine whether the received operations of the atom can be performed (or have been performed) and reply appropriately

Comment:

This message can be piggy-backed on an application message (in which case the operations of that message are referred to, as well as any prior messages for that atom).

PREPARE need not be sent to participants from whom VOTE has been received.

VOTE

Sent from participant to coordinator, either unsolicited or in response to PREPARE.

Parameter		
Atom identifier		
Vote	See below	
Timeout	applicable in the assume commit, assume rollback cases	
Application data	Can provide reasons for cancel or qualification of the ready message	

Vote	**Meaning**
cancel	the operations cannot be performed and the effects have been undone; the atom is no longer known to this participant
ready	the operations can be confirmed and can be cancelled, as may be instructed by the coordinator. The level of isolation is a local matter (i.e. is the participants choice, as constrained by the contract) – other access may be blocked, may see applied results of operation or may see original state (or cancelled)
ready, will assume confirm	as ready, but will lose the ability to cancel after the timeout
ready, will assume	as ready, but will automatically cancel after the timeout

cancel	

VOTE may be associated with an application message – typically the application response to a message associated with a prepare. In this case (for VOTE/ready) the application data field would normally be empty.

CONFIRM

Sent to a participant from whom VOTE with one of the ready results has been received

Parameter		
Atom identifier		

Meaning :

The atom is confirmed. The participant is released from the obligation to reverse the operations of the atom. The effects of the operation can be made available to everyone (if they weren't already)

No further messages for the atom will be sent, apart from resending the confirm in recovery.

CONFIRMED

Meaning:

The confirm has been applied

This message is really only needed to make the recovery logging work.

CANCEL

Sent to a participant at any time before (and unless) CONFIRM has been sent.

Parameter		
Atom identifier		

Meaning:

The atom is cancelled. The countereffects of any operations should be applied. The participant is released from the obligation to confirm the operations.

No further messages for the atom will be sent, apart from any resending in recovery.

CANCELLED

Sent by participant in reply to CANCEL.

Meaning:

The cancellation has been applied

This message is really only needed to make the recovery logging work.

COORDINATOR_STATUS

Sent by coordinator at any time, when it is (for whatever reason) uncertain whether the participant received the last message or, equivalently, the coordinator received no reply. Also sent in response to a received PARTICIPANT_STATUS, in particular states.

Parameter Atom identifier		
Status	See below	

Status	Meaning / Previous message sent/received
Active Preparing	ENROLL received PREPARE sent
Confirming	CONFIRM sent
Cancelling	CANCEL sent
Inaccessible	the atom may or may not be known, but the status cannot be determined at the moment
Unknown	the atom is not known: this implies the atom is cancelled.

Meaning:

Informs the participant of the current status of the coordinator. For status Preparing, Confirming and Cancelling, this effectively repeats the last message sent and the participant is to reply appropriately (possibly repeating a lost message from the participant). Status Active, Inaccessible and Unknown are only sent in response to a received PARTICIPANT_STATUS message.

A participant should always reply immediately to a received COORDINATOR_STATUS with Status Preparing, Confirming or Cancelling, using PARTICIPANT_STATUS only if none of the other messages are appropriate. (In particular, if the participant is ready, it should resend the VOTE)

Unknown must not be sent unless it has been determined for certain that the coordinator does not exist any more and will not exist. If there could be persistent information corresponding to the coordinator, but it is not accessible from the entity receiving the PARTICIPANT_STATUS message (or the entity cannot determine whether any such persistent information exists), the response must be Inaccessible.

PARTICIPANT_STATUS

Sent by participant at any time, when it is (for whatever reason) uncertain of the state of the atom as known to the coordinator. Also sent in response to a received COORDINATOR_STATUS, in particular states.

Parameter	
Atom identifier	
Status	See below
Timeout	applicable only in the assume commit, assume rollback cases

Status	**Meaning / Previous message sent/received**
Active Ready	ENROLL sent VOTE sent
Ready, assume confirm	VOTE sent
Ready, assume cancel	VOTE sent
Inaccessible	the atom may or may not be known, but the status cannot be determined at the moment
Unknown	the atom is not known; this implies the previous termination message (CANCEL or CONFIRM) did get through and was replied to

Meaning:

Informs the coordinator of the current status of the participant. For status Ready (and its variations), this effectively repeats the VOTE message. Status Active, Inaccessible and Unknown are only sent in response to a received COORDINATOR_STATUS message.

Unknown must not be sent unless it has been determined for certain that the participant does not exist any more and will not exist. If there could be persistent information corresponding to the participant, but it is not accessible from the entity receiving the COORDINATOR_STATUS message (or the entity cannot determine whether any such persistent information exists), the response must be Inaccessible.

A COORDINATOR_STATUS/PARTICIPANT_STATUS exchange that determines that one or both sides are in the active state does not require that the atom be cancelled (unlike some other two-phase commit protocols). The atom may be continued, with new application messages carrying the same context. Similarly, if the participant is ready but the coordinator is active, there is no required impact on the progression of the atom.

REDIRECT

Sent from either side when the address it has previously offered is no longer valid

Parameter	
Atom identifier	
Old address	The original address, now replaced
New address	The new address

Meaning:

> The entity (coordinator or participant) sending the message should now be accessed using the new address, not the old one. Access to the new address (especially the *_STATUS messages) should be able to return

This may occur in various circumstances, mostly involving a failure.

> Note: There is some interaction between redirection and the status query exchange.